A UNITED STATES
DEPARTMENT OF
COMMERCE
PUBLICATION

NBS TECHNICAL NOTE 797

Static Language Analysis

U.S.
DEPARTMENT
OF
COMMERCE
National
Bureau
of
Standards

NATIONAL BUREAU OF STANDARDS

The National Bureau of Standards[1] was established by an act of Congress March 3, 1901. The Bureau's overall goal is to strengthen and advance the Nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research and provides: (1) a basis for the Nation's physical measurement system, (2) scientific and technological services for industry and government, (3) a technical basis for equity in trade, and (4) technical services to promote public safety. The Bureau consists of the Institute for Basic Standards, the Institute for Materials Research, the Institute for Applied Technology, the Institute for Computer Sciences and Technology, and the Office for Information Programs.

THE INSTITUTE FOR BASIC STANDARDS provides the central basis within the United States of a complete and consistent system of physical measurement; coordinates that system with measurement systems of other nations; and furnishes essential services leading to accurate and uniform physical measurements throughout the Nation's scientific community, industry, and commerce. The Institute consists of a Center for Radiation Research, an Office of Measurement Services and the following divisions:

Applied Mathematics — Electricity — Mechanics — Heat — Optical Physics — Nuclear Sciences[2] — Applied Radiation[2] — Quantum Electronics[3] — Electromagnetics[3] — Time and Frequency[3] — Laboratory Astrophysics[3] — Cryogenics[3].

THE INSTITUTE FOR MATERIALS RESEARCH conducts materials research leading to improved methods of measurement, standards, and data on the properties of well-characterized materials needed by industry, commerce, educational institutions, and Government; provides advisory and research services to other Government agencies; and develops, produces, and distributes standard reference materials. The Institute consists of the Office of Standard Reference Materials and the following divisions:

Analytical Chemistry — Polymers — Metallurgy — Inorganic Materials — Reactor Radiation — Physical Chemistry.

THE INSTITUTE FOR APPLIED TECHNOLOGY provides technical services to promote the use of available technology and to facilitate technological innovation in industry and Government; cooperates with public and private organizations leading to the development of technological standards (including mandatory safety standards), codes and methods of test; and provides technical advice and services to Government agencies upon request. The Institute consists of a Center for Building Technology and the following divisions and offices:

Engineering and Product Standards — Weights and Measures — Invention and Innovation — Product Evaluation Technology — Electronic Technology — Technical Analysis — Measurement Engineering — Structures, Materials, and Life Safety[4] — Building Environment[4] — Technical Evaluation and Application[4] — Fire Technology.

THE INSTITUTE FOR COMPUTER SCIENCES AND TECHNOLOGY conducts research and provides technical services designed to aid Government agencies in improving cost effectiveness in the conduct of their programs through the selection, acquisition, and effective utilization of automatic data processing equipment; and serves as the principal focus within the executive branch for the development of Federal standards for automatic data processing equipment, techniques, and computer languages. The Center consists of the following offices and divisions:

Information Processing Standards — Computer Information — Computer Services — Systems Development — Information Processing Technology.

THE OFFICE FOR INFORMATION PROGRAMS promotes optimum dissemination and accessibility of scientific information generated within NBS and other agencies of the Federal Government; promotes the development of the National Standard Reference Data System and a system of information analysis centers dealing with the broader aspects of the National Measurement System; provides appropriate services to ensure that the NBS staff has optimum accessibility to the scientific information of the world. The Office consists of the following organizational units:

Office of Standard Reference Data — Office of Technical Information and Publications — Library — Office of International Relations.

---

[1] Headquarters and Laboratories at Gaithersburg, Maryland, unless otherwise noted; mailing address Washington, D.C. 20234.
[2] Part of the Center for Radiation Research.
[3] Located at Boulder, Colorado 80302.
[4] Part of the Center for Building Technology.

Static Language Analysis

Gordon Lyon

Systems and Software Division
Institute for Computer Sciences and Technology
U.S. National Bureau of Standards
Washington, D.C. 20234

t. Technical note no. 797

U.S. DEPARTMENT OF COMMERCE, Frederick B. Dent, *Secretary*
NATIONAL BUREAU OF STANDARDS, Richard W. Roberts, *Director*

Issued October 1973

National Bureau of Standards Technical Note 797

Nat. Bur. Stand. (U.S.), Tech. Note 797, 23 pages (Oct. 1973)
CODEN: NBTNAE

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402
(Order by SD Catalog No. C13.46:797). Price 50 cents.

CONTENTS

Static Language Analysis

Gordon Lyon

Although many variants of programming languages exist, little information is available on how language features are actually used by programmers. Several data collection schemes are discussed here; each would provide empirical data on language use. Some internal details are given for analyzers for FORTRAN and COBOL. In addition, a suggestion is made for a special systems option which would allow a compiler to continuously record source statement characteristics of programs given to it.

Key words: Data archives; language use; programming aids; programming languages; source-statement analysis; syntax analysis.

1. INTRODUCTION

Despite a proliferation of computer languages in the last decade, little has been gathered on actual, everyday use of programming languages and their compilers. Knuth[1.4] provides rudiments of what such studies might comprise. Professional language and compiler designers may scoff at an empirical approach to what has been essentially a realm of logic and axiomatics. Yet Knuth himself indicates that his personal experience included an overemphasis on clever design optimizations which would be invoked quite infrequently (according to his study).

The purpose of this report is to lay ground work for a reasonable language utilization analysis package. Such a utility would provide performance records to managers and language development groups. Statistics would be available on a Government-wide basis to measure processes of creation and maintenance of computer software. Analysis of programs could provide programmers with checkout and documentation aids, managers with maintenance and production statistics, and systems designers with utilization data.

1.1 Definition of Terms

Static analysis denotes a source-statement examination applicable to some class of programs. All FORTRAN programs -- correct or with errors -- typify such a class. Static analysis does not include features which interactive systems might provide. Specifically, no prompting is given. Objectives are thus for batch-run, program analyses.

Dynamic analysis denotes a statistical examination of a program during execution. This report does not discuss any aspect of dynamic analysis.

2. STATIC ANALYSIS

It is not difficult to give arguments on why managers, a standards committee, or NBS might want archives data on program language use: to determine representative statement mixes for compiler benchmarks; to design compact languages; to determine programmer performances; to reduce language-induced errors; to monitor the status of shop programs.

2.1 Possibilities

Discussion begins with an examination of three possible forms of static analysis data collection. The methods are: taps on streams of system compilers (a sleuth); programming aids which also record language-use data (a Trojan horse); final product auditor (the enforcer). A chart (cf. Table I) summarizes estimates of performance, costs, and audience for these three. Questions which might be asked of each method are:

1. Integrity of measurements. Do data from static analyses represent a good sampling of the population of programs? (A, C, below)

2. Archive adequacy. Do data contain answers to a wide collection of questions? (C, G)

3. Audience. What costs and effort do those using the end product justify? (B, D, E, F, G)

With items 1-3 above in mind, one can begin examining technical features of each method of static analysis.

2.1.1 Compiler Monitor. Best among analyzers in regard to collecting representative data, a compiler monitor is basically just a tap on some compiler input or output stream. In one instance, all daily input might be saved additionally on tape. If a thousand programs of 1000 cards each are compiled per day, the data may occupy over 15 reels of magnetic tape. Given the vital role that a compiler plays in an operational system, mechanisms to collect test data should interfere as little as possible with compiler executions. This factor may demand highly tailored assembly language interfaces, with their concommitant high programming cost. Furthermore, little time should be spent transforming data from the compiler before recording it, so as to limit overhead. The net result of the preceding arguments: the monitor may record huge volumes of data, tying up a complete tape drive and many reels of tape. Each day all these reels must then be reduced and merged to a master file. Although a monitor provides an excellent data base, great care must be exercised so that the data collection does not swamp operations.

	TAPS ON STREAMS OF SYSTEM COMPILER	PROGRAMMING AIDS WHICH RECORD LANGUAGE-USE DATA		FINAL PRODUCT AUDITOR
		simple	elaborate	
A. Collection Volume	(high) data from every program submitted to the compiler in question	(low) e.g. may only supply cross-references, thereby limiting data to jobs needing it	(medium-high) with good error messages, might get all submissions until free of error	(low) all finished programs
B. Development Costs Estimate	(low-high) tailored to each compiler and system; also depends upon how elaborate	(low) lexical level analyses, no syntactical data	(med.-high) lexical and syntactical; must be fairly efficient to run	(low-high) any range possible
C. Archives Reliability	(excellent)	(low?) vagaries in sampling process	(medium)	(medium) no data on errors available
D. Costs Above Normal	(more) each compilation has additional overhead	(extra) an additional run	(savings?) might replace first compilations if adequate syntax examination	(extra) an additional run
E. Installation Ease	(easy-difficult) catalogued procedures-easy; built-in command structure-hard. Also depends on compiler.	(easy) just another program	(easy)	(easy)
F. Type Audience	(system people, management) depends on what is collected	(programmers, systems types, management) depends		(management, systems)
G. Comments	Could provide good mixes for benchmark tests of compilers. Data perfectly representative. Overkill?	Variant of Knuth's work, also Young's. Inflexible, limited. Data base somewhat shaky done this way	Hosts more possible questions, has more users. Has error data, but column one prob. better	Management must enforce use of package. Data base valid for finished programs

TABLE I

Comparison of Three Methods of Static Analysis

2.1.2 Final Product Auditor. Simplest among the suggested methods, the product auditor assays completed programs. It depends heavily upon management diligence in requiring its use. Data acquired are representative, but contain no information on program errors, or other evolutionary aspects of program development.

2.1.3 Programming Aids Which Record. Representative of a middle ground, programming aids are proposed here as mutually beneficial Trojan horses. Programmers find them useful, but the intended aids also record systems data on language use. Because the simply conceived version is quite similar to a more elaborate programming aid, only the latter is developed here. Furthermore internal mechanisms (i.e. features other than programming aids) are common to all three methods of static analysis. Because these internals are shared by all methods, they are detailed in some degree.

2.2 Proposed Static Analysis Approach

Envision the static analyzer (TSA) as an extended compiler without code generation facilities. TSA is characterized by an external appearance as a programmer's aid. Internal functions of TSA gather systems information about programs and language.

Programming aids provide external features which entice programmers to use TSA. Wide usage is quite important to guarantee that archives are built upon representative, day-to-day programs, and not just polished end products. Without representative input, data on errors is useless. TSA should provide a cheap useful checkout of a program prior to compilation. This provides a natural, representative population for TSA to sample.

Besides providing aid to programmers, TSA records language use characteristics, frequencies, and programmer performance. These latter functions are transparent to programmers who use TSA. Such data as the analyzer collects will support archives on language use, programming behavior, host machine compatibilities, and representative mixes of language statements.

2.2.1 Visible Features. The visible features of TSA are aimed at an audience of actual language users. If these be FORTRAN programmers, then the following services have proven saleable with varying degrees of success: automatic statement label renumbering; neatening of decks; syntax checking; flowcharting; cross referencing; standards testing. Cross reference tables are most popular, probably because they are useful in debugging and documentation.

Flowcharts are useful for documentation, although opinions vary on their value. Statement label renumbering is useful in

FORTRAN, although not in COBOL. As a source statement check-out, TSA should lighten CPU loads and give faster turnaround. Student compilers operate on a similar philosophy. TSA includes many easily-specified options; each is, in effect, a carrot to lure programmers to TSA.

TSA could also aid documentation and standardization by flagging non-standard idioms, by indicating equivalents, and by indicating independent segments of code. Such services, designed to enhance program transferability, would be yet another option.

Equivalence maps are useful in FORTRAN programming, as are other indications of structural linkages with side effects. The IBM Fortran-H compiler carries this quite far, providing semantic assessments, detecting unused assignments, and moving constants out of loops.

COBOL services should differ from those for FORTRAN. There is more emphasis on standards - either American National Standard, FIPS, or ad hoc - and test data generation seems popular. Successful analyzers (cf. TABLE II) provide syntax checking, flow charting and cross-referencing.

One might wonder whether a source-scanner such as TSA would have any natural audience. Experience with FORTRAN -G and -H compilers indicates that programmers often run with the H-compiler not to optimize code, but to get symbol cross-reference tables! Furthermore, it is very convenient to have an additional error checkout of a program, since all compilers have their own weaknesses in syntax checking.

2.2.2 Transparent Features. TSA does not exist merely to expedite programming. Its utilitarian functions are chosen to encourage natural and widespread use.

Data collected by TSA should be useful for group studies of programmer performance. Program evolution could be studied, and one could extract data on language features vis-a-vis source errors. One problem is that observations are conditioned by abstract algorithms being programmed. These abstract algorithms could pose quite a problem, since observed program features vary drastically among algorithms.

TSA statistics provide a basis for language design. A principal difficulty here is that negligibly-used features are easily spotted, but missing "macros" will require some feature extraction by the analyzer. TSA might extract definable "macro" features, including those that are desirable based on error syndromes and frequencies. This information should aid language and compiler designers.

CODED IDENTIFICATION	RENUMBER	NEATEN	SYNTAX CHECK	FLOWCHART	CROSSREEFS	MACROS	STANDARDS	TEST DATA GEN	COMMENTS	
F1	o	o	o	o	o		o		source = FORTRAN	FORTRAN
F2				o	o				FORTRAN + assembly	FORTRAN
F3					?					FORTRAN
F4					o				multiple X-refs	FORTRAN
F5				o	o					FORTRAN
F6					o				FORTRAN deck, feeble	FORTRAN
F7				o	o					FORTRAN
C1			o	o	o					COBOL
C2 (F8)			o	o	o				many users	COBOL
C3						o				COBOL
C4		o				o	o			COBOL
C5				o	o	o				COBOL
C6				o					limited	COBOL
C7								o		COBOL
C8		o				o			source = COBOL	COBOL
C9						o			report generator	COBOL
C10			o				o			COBOL
C11				->					post-compiler, well liked	COBOL
C12		?							DATA DIVISION layouts	COBOL
C13							o		issues monthly reports	COBOL
C14							o	o	also run-time analysis	COBOL
C15							o		simple but useful	COBOL
C16			o		o				also run-time analysis	COBOL

Table II A Sampling of Static Analyzers (Commercial)

Frequencies in TSA archives could be used to determine statement and program "mixes" for a particular higher-level language. Truly representative benchmarks for specific language/machine combinations should then be possible, to aid evaluation of execution performance of various compilers.

American National Standards deviation could be evaluated from data on many compilers. A study of deviations may provide clues to current standard weaknesses.

2.3 Programming Aids and Services

Among aids available from commercial packages (cf. Figure 1 and TABLE II), cross reference tables are most popular. Syntax checking is another favorite, and standards a third.

The above dovetail nicely with transparent functions one might expect in systems sections of TSA. (Systems features are discussed shortly).

Figure 1 presents a hierarchy of TSA aids to programmers, along with indications of effort needed to realize a feature. Starting from the top node, features (in boxes) are realized by incorporating into TSA capabilities labeled on arcs. For example, cross references imply a language grammar and a symbol table. Longer paths have concommitantly more work associated with them. Solid lines (_____) indicate what seem to be reasonable visible features for FORTRAN. A dashed line is for COBOL. The reader may want to draw his own version of the areas after examining TABLE II. Relations in Figure 1 are only approximate.

More difficult programmer aids, such as flowcharting, have been excluded since they entail a good amount of effort (and overhead) on TSAs part. Use of such powerful services is probably limited to final versions of programs. This is not the population of programs TSA is designed to examine, since very little can be learned about everyday programming language use, difficulties, or programmer performance from a single, error-free deck. E. A. Youngs has argued this[1].

The following features seem useful as programming aids:

FORTRAN	COBOL
error analysis	neater deck
standards	shorthand expansion
cross references	error analysis
storage (interaction) map	standards
	cross references
	storage map

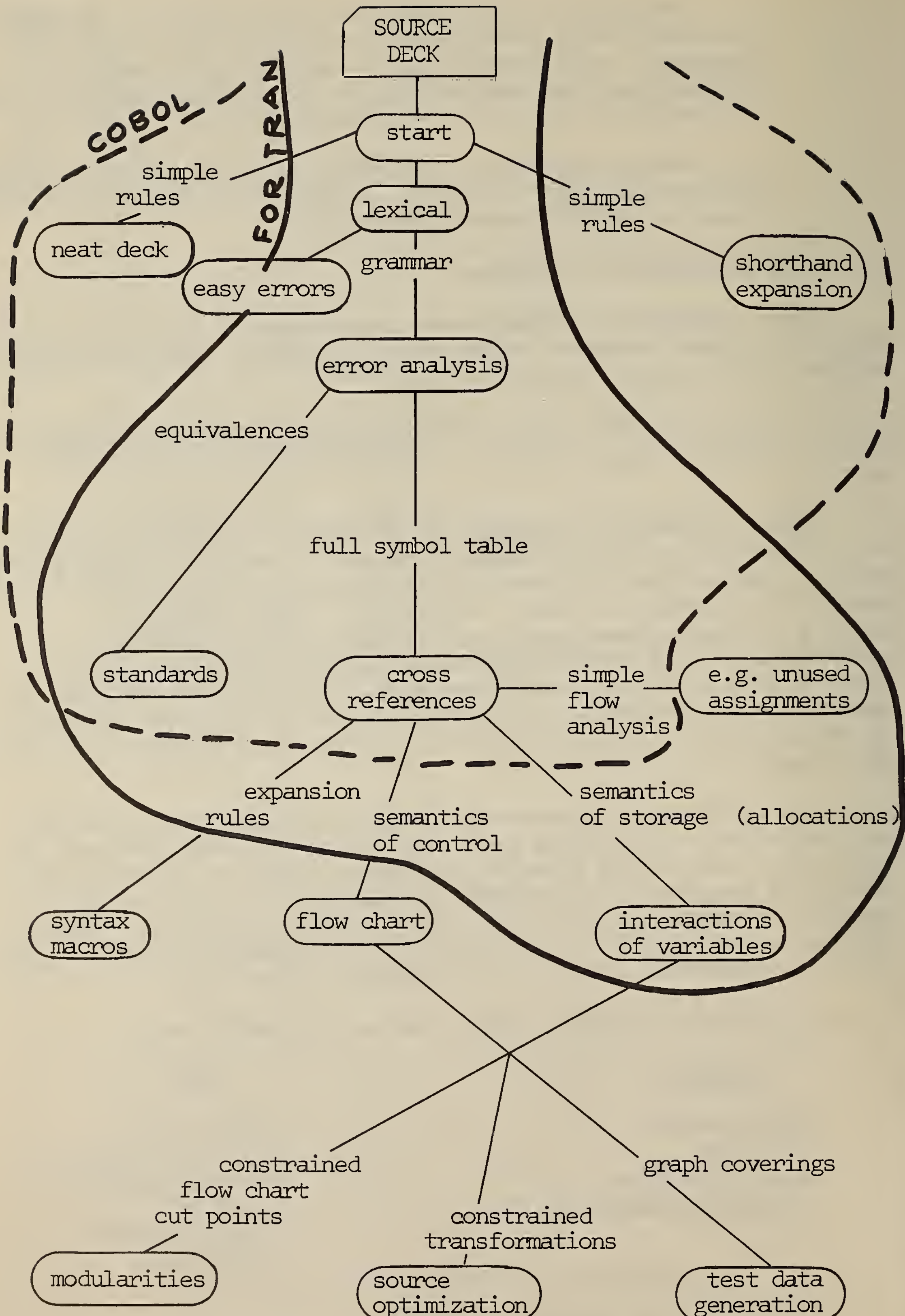


FIGURE 1 A possible hierarchy in features

2.4 Internal Features of TSA

True rationales for TSA are transparent to programmers, as bees are oblivious to pollen. The following seem worthy of study using TSA:

1. Programmer versus language: Program evolution, types of errors. This would be an attempt to automate a little of Youngs' work: "...We may find, for instance, that the iteration mechanism of PL/1 is more likely to contain programming errors than its FORTRAN counterpart. A possible source of this difference might be the FORTRAN requirement that a DO statement carry with it the label of its terminating line ... Quantified descriptions of such findings may be helpful in further developing the successors to FORTRAN and PL/1 for less error prone programming"[2].

2. Language use. Knuth has documented the frequencies of statement types in FORTRAN programs. Unfortunately, no published data exist for COBOL. Occurrence frequencies for both languages allow more efficient compiler design, as well as indicating (with judicious interpretation) utilities of statement types.

2.4.1 Specific TSA Features.

At a bare minimum TSA must perform a lexical analysis. And as an aid to programmers, TSA is enhanced if it can do syntactic analysis. A symbol table is necessary to account for language specification not covered by strict syntax alone.

If questions on program flow structure are asked, analysis must have a sufficient depth to answer them. Flow analyses are fairly time consuming. This precludes them from everyday use, except possibly for simple cases.

Internal features and external (programmer's aid) functions should dovetail. If an external facet requires syntax analysis, internal mechanism may as well assume the feature too, since the package must support it. Hence common algorithms support both internal and external features.

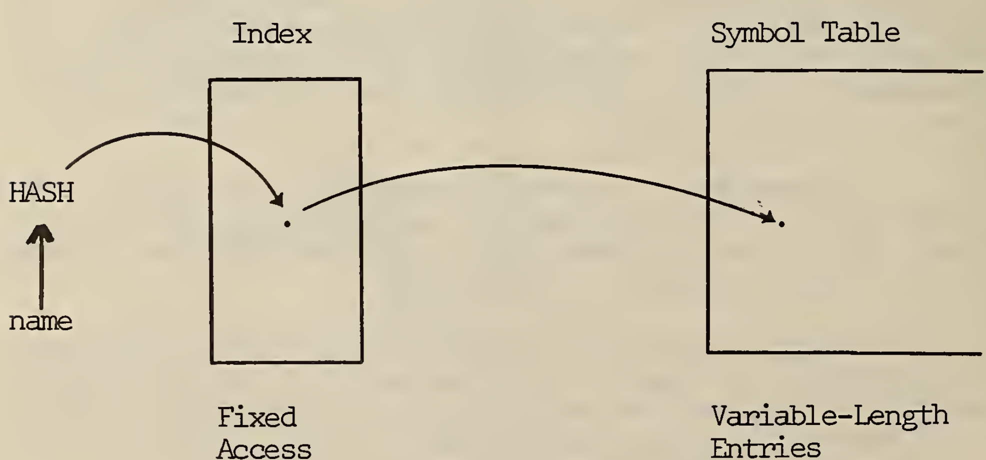
One very real question that arises is the difficulty in appropriately averaging the results of detailed syntactic analysis. With many distinct instances, a mean statistic may convey very little. Lexical elements of an arithmetic expression B*C+D are identical to those of B+C*D, although the respective parses are distinct. A simple comparison in types and numbers of operators requires no parsing information, and may be all that remains from an attempt to average results of parses. Knuth in one case chose a simple weighted counting "measure". The thrust of averaging is against a detailed

syntactic analysis, which in turn conflicts with requirements of a programmer's aid.

Internal structures of TSA should be extensible. This is a consequent of the range of questions which users might want to ask, the aggregate being too large for any one version of TSA. Extensibility need not be syntax-directed, but rather, consist of hooks and traps which programmers can fiddle with easily. Specific demands on implementation are:

1. Flexible and extensible tables.

As experimenters add and change attributes of items, tables must expand and implementation should allow this. One simple solution follows. Each symbol table entry for a particular application of TSA would be of fixed size. Entry size could be changed as each TSA application (or installation) required. Table entries would be accessed through a hash mechanism into a scatter index table. Once established, the index should not require modification, since allowed names in a language usually do not change in definition across individual programs written in it.



A garbage collection would not be necessary, since no items are freed in the symbol table.

2. Visible Access Methods

Tables should be accessed via one standard routine. Such a convention facilitates tabulating accesses for various

attributes. Changes to tabulation mechanisms are more consistent if done only in one place. A similar argument holds for syntactic portions of analysis. For instance, only one routine should be called to recognize variables. Any data on variables is available by monitoring this routine and its accesses to the symbol table.

In some cases TSA implementation will vary because of actual language syntax. This is especially true for a language such as COBOL. For instance, the construction

```
                                           { word-2       }
COPY  lib   [ REPLACING    word-1     BY   { identifier-1 }
                                           { literal-1    }

              [                { word-4       } ]     ]
              [ , word-3   BY  { identifier-2 } ] ... ]
              [                { literal-2    } ]     ]
```

appears in at least 11 places (!) in the COBOL definition. If COBOL's metalanguage allowed names to constructs, only one occurrence would be necessary, below:

```
<copy-lib >::=  COPY lib   [ REPLACING  word-1  ... etc. ]
```

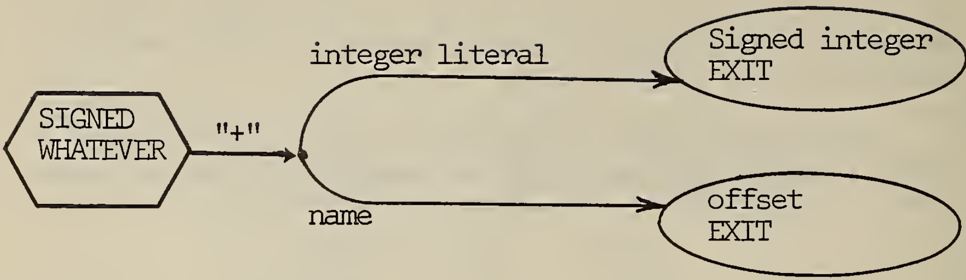
In any case, only one routine (or table) is required to analyze

COPY lib... constructs.

3. Syntax Analysis Should Proceed Without Backup.

This is important for efficiency and error handling. FORTRAN is easier to handle than COBOL. Conway's paper [5] provides an informal discussion of how one proceeds on COBOL. Tixier[6] and Lomet[15] have formalized Conway's results. Basically, one does analysis with transition diagrams with vari-

able returns, as below:



The diagrams for COBOL are, of course, recursive. Some care must be exercised to prevent backup conditions during COBOL analysis.

Programs submitted to TSA will have errors in them, and some decent error recovery mechanism must be incorporated into the syntactic analysis. Error diagnostics should be accurate and precise. A number of recent papers[7,8,9,11,12] discuss aspects of automatic error recovery and correction, but none of these techniques seem directly applicable to FORTRAN or COBOL analysis. Ad hoc methods along lines of Evans[10] may prove useful. A general strategy with transition diagrams is to jump to the next higher diagram upon error, and then scan input until context is suitable for resumption of the higher diagrams.

Many ad hoc techniques are available to break error recovery into smaller, more amenable pieces. Statement and sentence delimiters can be used as fiducial markers which partition a program. An important requirement in error recovery or correction is that errors should not appreciably slow the analysis.

4. Modular (structured; functional) Implementation.

A temporal, flow chart organization for TSA would not be convenient. Functional aspects should be isolated into distinct procedures. Since questions addressed to TSA are likely to be along functional lines, TSA structure will reflect this and be easier to modify. The example in 2.(above) illustrates virtues of modularity.

5. Some Semantics

It is convenient if TSA indicates storage allocations clearly. This requires analysis of symbol equivalences along lines of Arden[4] (FORTRAN) or Conway[5] (COBOL). The data can be displayed in a cross reference table, or in some more elaborate interaction display.

Example. For the following FORTRAN variables,

```
REAL B(30), INTEGER*2 A

DIMENSION K(13)

EQUIVALENCE (B(5), A), (A,K(1))
```

display of this form could be provided:

```
B(1)...B(5) B(6)...B(13)...B(30)

            .      .      .
            .      .      .
            .      .      .
            A
            K(1) K(2)...K(13)
```

2.4.2 Further Details for FORTRAN. As an example of mechanisms necessary for static analysis, consider the following instances for FORTRAN.

DOMAIN	QUESTIONS	SOURCE OF ANSWER
Names, variables	% types;%LHS, RHS*	symbol table,
COMMON	% of vars in it, avg. size of blocks, use of labels	lexical scan and common-processing routines. NOTE: will interact with EQUIVALENCE
EQUIVALENCE	% vars, size classes # of refs in pgm.	lexical scan, equivalence analysis routine, symbol table.
IF	types, consequents	syntax analysis
GOTO	direct?assigned?fwd? bkwd?avg. length, avg. # crossovers	symbol table processing, must sort some entries
DO	length, and nesting depth	scan symbol table using stack
arithmetic expressions	structures, operator mixes, etc.	syntax analysis

*RHS =(on)right hand side, LHS = left hand side

Information for keywords is stored in a fixed keyword symbol table. Questions related to keywords are answered via manipulations of entries to the table. Consider "crossovers" for GOTO. From the GOTO entry are chained pairs of start and target addresses SA/TA for each GOTO. Assigned and computed

GOTOs represent multiple entries (see below), one for each target address.

GOTO SA(1)/TA(1)--SA(2)/TA(2)--SA(3)/TA(3)TA´(3)TA"(3)--...

Crossover is computed by determining an interval SA(i)/TA(i), checking for other GOTOs such that SA(j) is outside of SA(i):TA(i) and TA(j) is in, or TA(j) is out and SA(j) in the interval.

Many other actions must be performed. For instance, DO entries with nesting:

DO SA(1)/TA(1)--SA(2)/TA(2)--SA(3)/TA(3)-..

can be counted by pushing TA(j)s onto a stack. DO entries (above) are sorted with ascending start address SA. If SA(j+1) <TA(j), then a nesting exists and TA(j+1) should be pushed onto the stack. A counter of stack depth is also incremented. Otherwise, all top-of-stack elements TA(.) <SA(j+1) are unstacked. The stack counter is decremented for each stack element removed. For any DO the stack counter gives a nesting depth.

Again, arithmetic statements present a problem. A measure of structure is necessary if statistics are to be gathered on them. If Knuth's approach is adequate, then statistics available from TSA will differ from his mostly in any information on errors.

FORTRAN analysis will begin with an ad hoc scan. Statements will be translated into a standard format, and their type indicated. Lexical and syntactic analysis will follow. It remains to be seen whether a single analyzer can efficiently check (simultaneously) for various standards, such as American National Standards. Individual tailoring may be necessary for an installation. Written in American National Standards FORTRAN, the package should be easy to experiment with.

2.4.3 An Approach to COBOL. COBOL usage is somewhat more difficult to monitor than FORTRAN. Because COBOL has so many independent options, data collection must be constructed carefully to avoid combinatorial problems. However, keeping in mind that only those characteristics of language which are invariant across many programs are of any value, satisfactory progress can be made with COBOL constructs. It is sensible, for example, to record numbers of statements, numbers of variables

per program, and average lengths of names. Recording specific names in a particular program is futile. Similarly, for a phrase with iteration options, such as

<x> = [,<identifier>]...

which has realizations of "NULL," "A,B", "A,B,C,D,E,F", etc., a good set of statistics is i) number of times that phrase <x> occurred and ii) average number of identifiers in each occurrence. Thus only two counters NPX (number of phrase <x>) and SPX (average size of phrase <x>) need be kept. For the recursive COBOL construct IF <β>; <statement>; ELSE <statement>, several counters might be kept for "IF", each corresponding to a level of nesting at which an "IF" is found.

A table of about 550 entries is adequate to record data on the composite language skeleton of COBOL (USA STANDARD COBOL, X3.23, pp. 1-102-1-117). Each entry can be updated via a recurrence relation, so it is necessary to keep only a small table of constantly updated archives data. Such a collection mechanism would be very useful if built into a compiler.

3. CONCLUSIONS

Among possible approaches to static analysis, the concept of the analyzer as a programmer's aid appears to benefit the largest audience. This version of TSA should have such technical features as: extensible internals; efficient syntax analysis; symbol table semantic checks. If developed very carefully, TSA might provide programmers with a very useful checkout tool.

Table II displays attributes of FORTRAN and COBOL static analyzers which are available on the open market. Entries in Table II are representative of products sold as programming aids, flow chart packages and documentation programs[16,17].

Although a catalog search is never conclusive, there seems to be a preponderance of COBOL programs in comparison to FORTRAN. And of all packages, only one (C_{13} in Table II) issues monthly systems reports on standards adherence, system application routine statuses, and the like. Other packages have features similar to TSA; this is hardly an accident.

None of the packages appear to satisfy TSA requirements directly. Few of the available systems are written in a source language compatible with the NBS Univac 1108 system, or even in any high level language. It is difficult to estimate whether a flowchart package is suited for modification to a TSA, since "error checking" is apparently rather rudimentary sometimes. Such a package would require both syntax analysis and symbol table additions. Not only would these constitute a major investment, but efficiency might suffer as a consequence.

3.1 Future Plans

Because FORTRAN is easier to handle, a prototype TSA will be written for it. A series of modules, one for each statement type, will decompose statements which have been processed previously by a preliminary scanner. Knuth[13] indicates that his scanner was a rather simple thing. Some error recovery will be included within the NBS development. Use of a TSA on FORTRAN will help determine whether such an approach is feasible and warranted for COBOL.

A FORTRAN analysis can be summarized in many different ways. Present plans are for TSA itself to compute summary statistics, such as GOTO crossovers. A more flexible alternative records selected symbol table entries from each analysis. This philosophy would entail a high capacity secondary store, such as magnetic tape. It should be an easy matter to convert from one collection strategy to the other if summary routines are given clean interfaces in TSA.

Design of an efficient COBOL syntax checker requires further study. In particular, a consistent set of transition diagrams must be developed, along with an interpreter for them. An attempt at obtaining consistent transition diagrams from outside sources has been rather disappointing.

A parallel effort has investigated a design for a simple keyword-compounds analyzer for COBOL. Only correct COBOL programs are analyzed. Counts of constructs such as DIVIDE, DIVIDE ROUNDED, etc. are built up. Such syntax analyses may provide a useful first view of COBOL use.

Yet another method that will be assessed is to use a command file in Infonet. Users would transfer control to a command file rather than invoke the COBOL compiler directly. Output would go to a file. Error messages could then be tallied and recorded in yet another file. Finally, appropriate file entries would be sent back to the terminal user. In this manner information could be gleaned about COBOL errors.

3.2 A Suggestion for Compiler Producers

The best method of recording language statistics is for compilers to tally them. Data on length of programs, frequencies of keywords, error messages and severities -- all these are handily available during compilation. It would be quite easy to design a compiler which kept a log of language feature utilizations. The extra cost to extend compilers to log data should be slight, yet the resultant data immensely useful.

4. REFERENCES

[1] Youngs, E.A., Error-proneness in programming. PH.D. thesis. University of North Carolina(1969).

[2] Idem, p.11.

[3] Morris, R. Scatter storage techniques. C.ACM 11, 1(January,1968), 38-44.

[4]. Arden, B.W., Galler, B.A., and Graham, R.M., An algorithm for equivalence declarations. C.ACM 4, 6(July, 1961), 310-314.

[5] Conway, M.E., Design of a separable transition-diagram compiler. C.ACM 6, 7(July, 1963), 396-408.

[6] Tixier, V., Recursive functions of regular expressions in language analysis. Ph.D. thesis. Stanford University (1967).

[7] Levy, J.P., Automatic correction of syntax errors in programming languages. Ph.D. thesis. Cornell University (1971).

[8] LaFrance, J.E., Syntax-directed recovery for compilers. Ph.D. thesis. University of Illinois at Urbana-Champaign (1971).

[9] Leinius, R.P., Error detection and recovery for syntax-directed compiler systems. Ph.D. thesis. University of Wisconsin (1970).

[10] Evans, A. Jr., An ALGOL 60 compiler. Annual Review in Automatic Programming. New York: Pergamon Press 1964.

[11] Lyon, G.E., Least-errors recognition of mutated context-free sentences in time $n^3 \log n$. Proc., Sixth Princeton Conf. on Information Systems and Sciences (March 1972). Also: A syntax-directed least-errors recognizer for context-free languages. Ph.D. thesis. University of Michigan (1972).

[12] Peterson, T.G., Syntax error detection, correction and recovery in parsers. Ph.D. thesis. Stevens Institute of Technology (1972).

[13] Knuth, D., Note to the author. April, 1973.

[14] Knuth, D., An empirical study of FORTRAN programs. Software-Practice and Experience 1. (1971), 105-133.

[15] Lomet, D. B., A formalization of transition diagram systems. C.ACM 20,2(April,1973), 235-257.

[16] Computer Technology Reports. pp 978-1435.020 to 980.6331-02 (Auerbach Publishers, Inc., Philadelphia, 1972).

[17] Programmer aids. ICP Quarterly, (Oct. 1972, Jan. 1973).

☆ U. S. GOVERNMENT PRINTING OFFICE : 1973—542-650/54

FORM NBS-114A (1-71)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET	1. PUBLICATION OR REPORT NO. NBS TN-797	2. Gov't Accession No.	3. Recipient's Accession No.
4. TITLE AND SUBTITLE Static Language Analysis			5. Publication Date October 1973
			6. Performing Organization Code
7. AUTHOR(S) Dr. Gordon Lyon			8. Performing Organization
9. PERFORMING ORGANIZATION NAME AND ADDRESS NATIONAL BUREAU OF STANDARDS DEPARTMENT OF COMMERCE WASHINGTON, D.C. 20234			10. Project/Task/Work Unit No.
			11. Contract/Grant No.
12. Sponsoring Organization Name and Address Same as No. 9			13. Type of Report & Period Covered Final
			14. Sponsoring Agency Code

15. SUPPLEMENTARY NOTES

16. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.)

Although many variants of programming languages exist, little information is available on how language features are actually used by programmers. Several data collection schemes are discussed here; each would provide empirical data on language use. Some internal details are given for analyzers for FORTRAN and COBOL. In addition, a suggestion is made for a special systems option which would allow a compiler to continuously record source statement characteristics of programs given to it.

17. KEY WORDS (Alphabetical order, separated by semicolons)
Data archives; language use; programming aids; programming languages; source-statement analysis; syntax analysis.

18. AVAILABILITY STATEMENT	19. SECURITY CLASS (THIS REPORT)	21. NO. OF PAGES
[X] UNLIMITED.	UNCLASSIFIED	23
[ ] FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NTIS.	20. SECURITY CLASS (THIS PAGE) UNCLASSIFIED	22. Price $.50

USCOMM-DC 66244-P71

NBS TECHNICAL PUBLICATIONS

PERIODICALS

JOURNAL OF RESEARCH reports National Bureau of Standards research and development in physics, mathematics, and chemistry. Comprehensive scientific papers give complete details of the work, including laboratory data, experimental procedures, and theoretical and mathematical analyses. Illustrated with photographs, drawings, and charts. Includes listings of other NBS papers as issued.

Published in two sections, available separately:

- **Physics and Chemistry (Section A)**

Papers of interest primarily to scientists working in these fields. This section covers a broad range of physical and chemical research, with major emphasis on standards of physical measurement, fundamental constants, and properties of matter. Issued six times a year. Annual subscription: Domestic, $17.00; Foreign, $21.25.

- **Mathematical Sciences (Section B)**

Studies and compilations designed mainly for the mathematician and theoretical physicist. Topics in mathematical statistics, theory of experiment design, numerical analysis, theoretical physics and chemistry, logical design and programming of computers and computer systems. Short numerical tables. Issued quarterly. Annual subscription: Domestic, $9.00; Foreign, $11.25.

DIMENSIONS, NBS

The best single source of information concerning the Bureau's measurement, research, developmental, cooperative, and publication activities, this monthly publication is designed for the layman and also for the industry-oriented individual whose daily work involves intimate contact with science and technology —*for engineers, chemists, physicists, research managers, product-development managers, and company executives.* Annual subscription: Domestic, $6.50; Foreign, $8.25.

NONPERIODICALS

Applied Mathematics Series. Mathematical tables, manuals, and studies.

Building Science Series. Research results, test methods, and performance criteria of building materials, components, systems, and structures.

Handbooks. Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications. Proceedings of NBS conferences, bibliographies, annual reports, wall charts, pamphlets, etc.

Monographs. Major contributions to the technical literature on various subjects related to the Bureau's scientific and technical activities.

National Standard Reference Data Series. NSRDS provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated.

Product Standards. Provide requirements for sizes, types, quality, and methods for testing various industrial products. These standards are developed cooperatively with interested Government and industry groups and provide the basis for common understanding of product characteristics for both buyers and sellers. Their use is voluntary.

Technical Notes. This series consists of communications and reports (covering both other-agency and NBS-sponsored work) of limited or transitory interest.

Federal Information Processing Standards Publications. This series is the official publication within the Federal Government for information on standards adopted and promulgated under the Public Law 89–306, and Bureau of the Budget Circular A–86 entitled, Standardization of Data Elements and Codes in Data Systems.

Consumer Information Series. Practical information, based on NBS research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

BIBLIOGRAPHIC SUBSCRIPTION SERVICES

The following current-awareness and literature-survey bibliographies are issued periodically by the Bureau:

Cryogenic Data Center Current Awareness Service (Publications and Reports of Interest in Cryogenics). A literature survey issued weekly. Annual subscription: Domestic, $20.00; foreign, $25.00.

Liquefied Natural Gas. A literature survey issued quarterly. Annual subscription: $20.00.

Superconducting Devices and Materials. A literature survey issued quarterly. Annual subscription: $20.00. Send subscription orders and remittances for the preceding bibliographic services to the U.S. Department of Commerce, National Technical Information Service, Springfield, Va. 22151.

Electromagnetic Metrology Current Awareness Service (Abstracts of Selected Articles on Measurement Techniques and Standards of Electromagnetic Quantities from D-C to Millimeter-Wave Frequencies). Issued monthly. Annual subscription: $100.00 (Special rates for multi-subscriptions). Send subscription order and remittance to the Electromagnetic Metrology Information Center, Electromagnetics Division, National Bureau of Standards, Boulder, Colo. 80302.

Order NBS publications (except Bibliographic Subscription Services) from: Superintendent of Documents, Government Printing Office, Washington, D.C. 20402.

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Washington, D.C. 20234

OFFICIAL BUSINESS

Penalty for Private Use, $300

POSTAGE AND FEES PAID
U.S. DEPARTMENT OF COMMERCE
COM-215

U.S.MAIL